

**PA3 Report**

By

**Grant Hedley**

**CSI 281**

**Data Structures & Algorithms**

**Fall 2019-20**

1. **Introduction**

The purpose of this experiment is to determine the efficiency of 6 different sorting algorithms (Bubble, Insertion, Selection, Shell, Merge, and Quick) by measuring how long each of them took to sort a dataset in microseconds.

1. **Background**

Bubble sort: this sort is the most basic, it will just run through a list comparing two adjacent items, and swapping them, until the list is sorted. Every cycle through the list will always put the greatest item at the end of the list so we can optimize the algorithm to only go through the list up to the last item we know is sorted. This is still the slowest sort in this list, even with the optimization taking about 45 minutes to sort a list of 1,000,000 random integers. However, not having any steps other than checking if all the elements are sorted, it is the fastest at confirming that a sorted list is already sorted.

Insertion sort: The principle of insertion sort is that it takes an element, then looks backwards down the list, and inserts it in the list before the first element that is smaller than it. Insertion sort is one of the faster sorts, taking about 9 minutes to sort 1,000,000 random integers but it takes twice as long in its worst case scenario.

Selection sort: This sort goes through each element and switches it with the smallest element in the list. Selection sorts most notable problem is it’s best case, average case, and worst case scenarios are all the same.

Shell sort: Shell sort works the same as bubble sort, but instead of comparing adjacent elements it compares elements separated by a gap. Each pass tish gap gets smaller, until there is no gap and it’s just a bubble sort.

Merge sort: This is one of two recursive algorithms in this list, and one of the two fastest algorithms. merge sort separates its elements into smaller lists then *merges* the smaller lists together so that the two (already sorted lists) become one larger sorted list.

Quick sort: This is the other recursive algorithm, and the fastest algorithm on this list, never taking more than 0.15 seconds to sort any list. It picks an element to be a “pivot” and goes through the list from the front and the back, putting each element that is bigger than the pivot on one side of the pivot and the smaller elements on the other. After each run through the pivot is sorted and it runs itself on the two halves it created on either side of the pivot.

1. **Implementation Detail**

All the algorithms were template functions in c++. Bubble sort was implemented as the optimized version I explained above. merge sort was written as 2 functions, a function to merge the lists together, and one to split up the lists and call the merge function.

1. **Experimentation Detail**

Each algorithm was run through 3 different data sets 3 times, each time with a larger portion of the dataset. Then this was repeated 3 times to get an idea of the average time it would take the algorithm to sort each list.

* 1. Memory on board: 7.44 G
  2. Processor type: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
  3. CPU speed: 3.9GHz-400MHz
  4. System type: 64 bits

**Summary Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: Bubble Sort** | | | |
| **N** | **Dataset #1[[1]](#footnote-0)** | **Dataset #2** | **Dataset #3** |
| 100 | 21.333μs | 0.333μs | 19μs |
| 10,000 | 222,542.333μs | 0.333μs | 178,368.333μs |
| 1,000,000 | 2,729,075,161μs | 2,200.666μs | 1,803,289,311.666μs |

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: InsertionSort** | | | |
| **N** | **Dataset #1[[2]](#footnote-1)** | **Dataset #2** | **Dataset #3** |
| 100 | 9.333μs | 0.666μs | 12.666μs |
| 10,000 | 55,702.666μs | 31.666μs | 111,503.666μs |
| 1,000,000 | 549,914,061.666μs | 3,210μs | 1,101,414,616.333μs |

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: MergeSort** | | | |
| **N** | **Dataset #1[[3]](#footnote-2)** | **Dataset #2** | **Dataset #3** |
| 100 | 11.666 | 7.333 | 8 |
| 10,000 | 1,462.333 | 894.333 | 936.333 |
| 1,000,000 | 20,5861.333 | 118617 | 122,496 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: QuickSort** | | | |
| **N** | **Dataset #1[[4]](#footnote-3)** | **Dataset #2** | **Dataset #3** |
| 100 | 6.333 | 3.666 | 2.333 |
| 10,000 | 934.666 | 331.333 | 366.333 |
| 1,000,000 | 130,297 | 41,446 | 41,733 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: SelectionSort** | | | |
| **N** | **Dataset #1[[5]](#footnote-4)** | **Dataset #2** | **Dataset #3** |
| 100 | 14 | 11 | 15 |
| 10,000 | 100886 | 100,225.333 | 149620 |
| 1,000,000 | 1,005,752,303.333 | 1,005,083,857 | 1,876,142,882.333 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: ShellSort** | | | |
| **N** | **Dataset #1[[6]](#footnote-5)** | **Dataset #2** | **Dataset #3** |
| 100 | 15 | 1.333 | 14.666 |
| 10,000 | 108,896.333 | 252.333 | 107105.333 |
| 1,000,000 | 1412372091 | 38,089 | 1,052,496,544.666 |

1. **Discussion and Conclusion**
2. Explain the results that you collected.
   1. The larger the list the longer the algorithms take to sort it.
   2. the sorted dataset was always the fastest (ignoring selection sort) and the worst case scenario was usually the slowest but there are some algorithms that benefit from the elements being in order even if its the wrong order, like shell sort.
3. These were the results I was expecting
4. quick sort, merge sort, insertion sort, shell sort, selection sort, bubble sort
5. My favorite algorithm is quick sort, because it’s so fast and it makes a lot of sense in my head. I love this algorithm so much that the other day I was in bed and started writing quicksort implementations in different languages on my phone.
6. **References**
7. **Appendix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Bubble Sort** | | | **Dataset #:** 1 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 21 | 20 | 23 | 21.333 |
| 10,000 | 21,5824 | 22,6160 | 22,5643 | 222,542.333 |
| 1,000,000 | 2,731,610,235 | 2,731,200,207 | 2,724,415,041 | 2,729,075,161 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Bubble Sort** | | | **Dataset #:** 2 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0 | 1 | 0 | 0.333 |
| 10,000 | 22 | 21 | 21 | 0.333 |
| 1,000,000 | 2,192 | 2,218 | 2,192 | 2,200.666 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Bubble Sort** | | | **Dataset #:** 3 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 19 | 19 | 19 | 19 |
| 10,000 | 178004 | 178877 | 178224 | 178,368.333 |
| 1,000,000 | 1806636027 | 1802120417 | 1801111491 | 1,803,289,311.666 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: InsertionSort** | | | **Dataset #:** 1 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 7 | 8 | 13 | 9.333 |
| 10,000 | 53721 | 56785 | 56602 | 55,702.666 |
| 1,000,000 | 548,870,678 | 550,748,621 | 550,122,886 | 549,914,061.666 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: InsertionSort** | | | **Dataset #:** 2 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 1 | 0 | 1 | 0.666 |
| 10,000 | 32 | 32 | 31 | 31.666 |
| 1,000,000 | 3,244 | 3,228 | 3,158 | 3,210 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: InsertionSort** | | | **Dataset #:** 3 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 13 | 12 | 12 | 12.666 |
| 10,000 | 112680 | 112307 | 109524 | 111,503.666 |
| 1,000,000 | 1,101,518,769 | 1,102,301,340 | 1,100,423,740 | 1,101,414,616.333 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: MergeSort** | | | **Dataset #:** 1 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 12 | 11 | 12 | 11.666 |
| 10,000 | 1468 | 1460 | 1459 | 1,462.333 |
| 1,000,000 | 20,5722 | 20,5525 | 20,6337 | 20,5861.333 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: MergeSort** | | | **Dataset #:** 2 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 7 | 7 | 8 | 7.333 |
| 10,000 | 897 | 892 | 894 | 894.333 |
| 1,000,000 | 118129 | 118679 | 119043 | 118617 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: MergeSort** | | | **Dataset #:** 3 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 8 | 8 | 8 | 8 |
| 10,000 | 935 | 936 | 938 | 936.333 |
| 1,000,000 | 122,010 | 122,690 | 122,788 | 122,496 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: QuickSort** | | | **Dataset #:** 1 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 6 | 6 | 7 | 6.333 |
| 10,000 | 934 | 935 | 935 | 934.666 |
| 1,000,000 | 129,411 | 131,179 | 130,301 | 130,297 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: QuickSort** | | | **Dataset #:** 2 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 4 | 4 | 3 | 3.666 |
| 10,000 | 316 | 363 | 315 | 331.333 |
| 1,000,000 | 41,451 | 41,460 | 41,427 | 41,446 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: QuickSort** | | | **Dataset #:** 3 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 2 | 3 | 2 | 2.333 |
| 10,000 | 419 | 318 | 362 | 366.333 |
| 1,000,000 | 41,751 | 41,740 | 41,708 | 41,733 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: SelectionSort** | | | **Dataset #:** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 14 | 14 | 14 | 14 |
| 10,000 | 100983 | 100908 | 100767 | 100886 |
| 1,000,000 | 1,005,746,570 | 1,004,964,358 | 1,006,545,982 | 1,005,752,303.333 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: SelectionSort** | | | **Dataset #:** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 11 | 11 | 11 | 11 |
| 10,000 | 100,182 | 100,180 | 100,314 | 100,225.333 |
| 1,000,000 | 1,005,243,407 | 1,004,820,519 | 1,005,187,645 | 1,005,083,857 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: SelectionSort** | | | **Dataset #:** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 14 | 15 | 15 | 15 |
| 10,000 | 150434 | 149195 | 149231 | 149620 |
| 1,000,000 | 1,879,661,964 | 1,873,418,626 | 1,875,348,057 | 1,876,142,882.333 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: ShellSort** | | | **Dataset #:** 1 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 16 | 13 | 16 | 15 |
| 10,000 | 105,528 | 98,633 | 122,528 | 108,896.333 |
| 1,000,000 | 1297458523 | 1463294647 | 1476363103 | 1412372091 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: ShellSort** | | | **Dataset #:** 2 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 1 | 1 | 2 | 1.333 |
| 10,000 | 252 | 252 | 253 | 252.333 |
| 1,000,000 | 38,047 | 38,016 | 38,204 | 38,089 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: ShellSort** | | | **Dataset #:** 3 | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 14 | 15 | 15 | 14.666 |
| 10,000 | 107173 | 107042 | 107101 | 107105.333 |
| 1,000,000 | 1,051,317,507 | 1,050,814,013 | 1,055,358,114 | 1,052,496,544.666 |

1. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-0)
2. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-1)
3. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-2)
4. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-3)
5. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-4)
6. Dataset #1 is average case, dataset #2 is best case and dataset #3 is worst case. [↑](#footnote-ref-5)